Visual Basic For Applications

(VBA) - Microsoft Excel Specific Primer

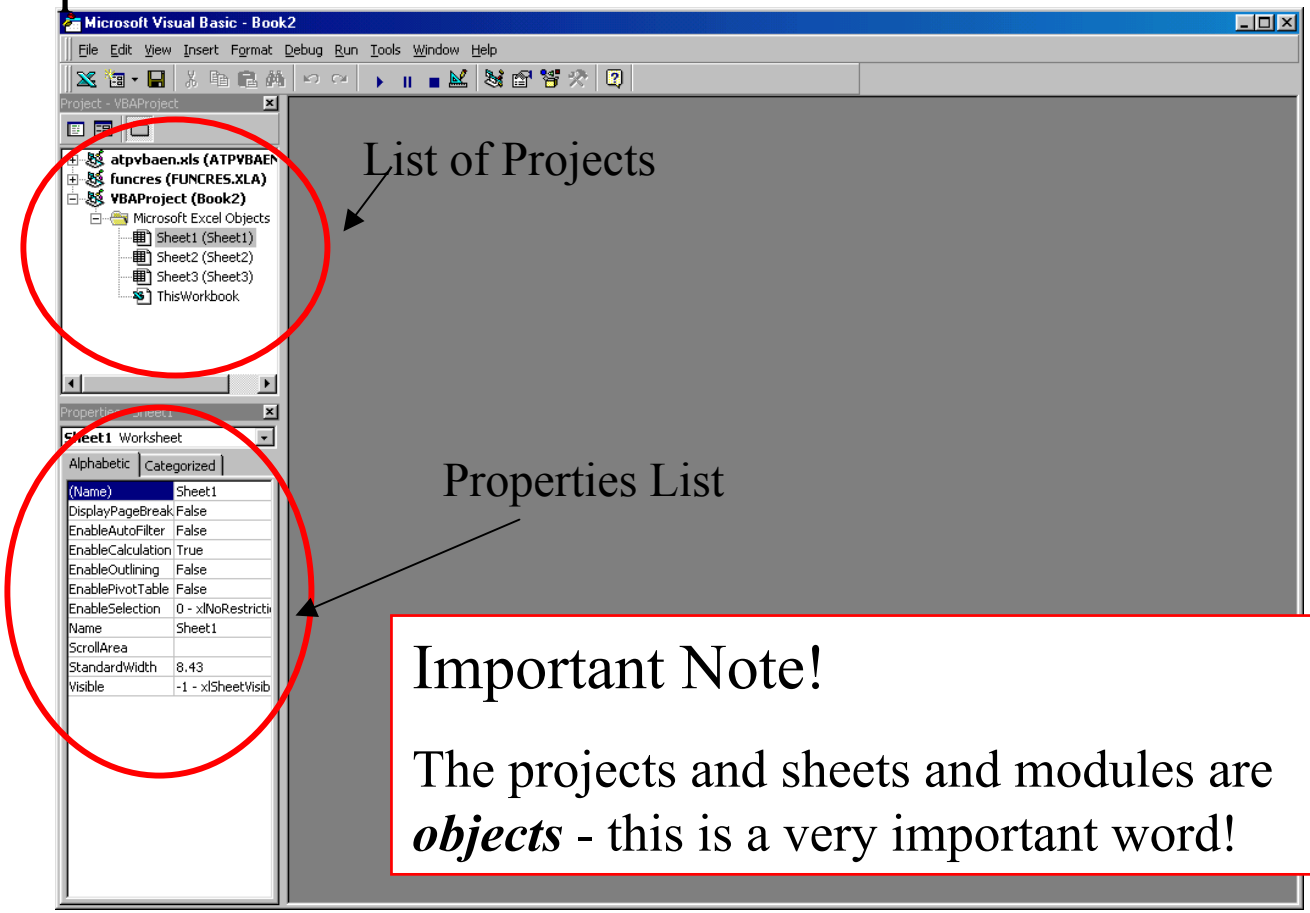
- This discussion IS NOT
  - going to teach you how to program
  - going to tell you every feature in VBA
  - about discussing the merits of algorithms
  - guaranteed to be politically correct
- This discussion IS
  - concentrated on how to make applications for END USERS who ARE NOT EXPERTS
  - predicated on the '*design* from the **top down**, but *implement* from the **bottom up**' philosophy
  - unabashedly biased and opinionated throughout

So what is this VBA stuff, anyway?

- A real programming language with a few, usually forgettable, distinctions from standalone Visual Basic
  - And one important one: *The program is embedded 'under' the spreadsheet and can not be run outside of Excel*
- A way to write Windows code without having to think about jargon B.S. like 'APIs' and 'threads' and such
  - Unfortunately, not fully standardized across Word, Excel, Powerpoint and Access (*especially* Access) -but the basics are the same.
- A way to use Excel as a *formatting and layout tool* so that coding effort is expended on *getting the job done, not building the graphical user interface.*

Nuts and Bolts - the tools you have

- Fire up Excel. Hit Alt-F11. This is the editor.



Important Note!

The projects and sheets and modules are ***objects*** - this is a very important word!

A First Subroutine

- Hit Tools…References. This lists a large number of code libraries you can 'add-in' to do lots of things. The default set will do for many things, but remember this - it will be important in later sessions.
- OK, let's create some code and hook it to something.
  - Cells in the worksheet, and the worksheet itself, are *objects* to manipulate. So let's write some code to fill cells A2..C7 with some values.

```
Option Explicit
Public Sub fill_some_cells()
    Worksheets("Sheet1").Range("A2:C7").Value = 6
End Sub
```

Simply double-click on Sheet1, type it in. Compile. Run from Excel.

Hook it to something

- Head back to Excel. Use Tools..Customize to turn on the **Controls Toolbox.**
  - Controls are *objects* (that is, things that have *properties* you can putz with and ways to *do something,* or "methods")
  - Turn on Design Mode (the little drafting triangle)
  - Make a button
  - Right click, and try 'view code'
- Hey, you're back in VBA! And look, you have a Module now.
- Let's play with the button a bit. It has properties we can play with.
- Note the effect of being in 'design' vs. 'run' mode.
- Examine the code side of things.
  - See all the different things in the right side drop down box? These are the various actions you can write code to respond to.

The Windows OS - always lurking in back

- Let's think about all these potential subroutines laying about. Note we've had some code do something, but there's not been any 'main program'. In Windows, it doesn't work that way.
  - Windows wants to be the 'main' for all programs at all times
  - Any subroutine in any program, no matter how complex, could be entered at any time
  - Interrupts are everywhere! If you click fast enough it's possible to execute subroutine 'b' *in the middle of executing subroutine 'a' and 'a' will never know about it!*
    - This can wreak real damage if 'a' and 'b' use the same variables!
  - The wise programmer will insure that critical things have a way to disable everything else until the critical operation is complete.
  - This structure means there are never any 'wait forever for user to click' loops

Let the machine do the work.

- OK, that's pretty trivial. What about a routine that clears cells A1..C9, then fills A1..A9 with values 1,2,3,..., then sets the B-row cells to the formula (Ax*5) and the C-column cells to the formula (Ax*Ax/6), then formats the C column cells so that they're red if the C value is greater than the B?
  - Lots of pain in the arse, right? ***Wrong.***
  - Go back to Excel. Hit Tools…Macro and *let Excel write the junky stuff for you.*
  - All the macros are captured as *Modules* in VBA, and you can export/import modules to text files for re-use.
  - Be forewarned - things are not always the same between versions of Excel! What works in Excel 97 might not in Excel 2000!

Understanding the Notation

- If you've ever programmed in C and used *structures* this will be very familiar. The concept and notation is not new; this notation technique has been around since the mid-70s.
- If you want to play with something in Excel, you simply say

Application("Excel").Worksheets("Sheet1").Cells(r,c).property = 'something'

to set a value associated with the lowest level object (the property) to 'something'

or

something = Application("Excel").Worksheets("Sheet1").Cells(r,c).property

to retrieve the value of the lowest-level object to variable 'something'

Note: often the first one or two levels are *assumed* or implicit using the **WITH** statement… can be good, often makes for unreadable code

What about Functions?

- Alright, we've built a *subroutine* - a chunk of code to do something with data. And, we've seen how it can manipulate the cells in the worksheet. How do you send information between routines?
- Functions. Try this one.

```
Option Explicit
Private Function jta1(A As Integer, B As Integer) As Double
   Dim temp1 As Double
   temp1 = (A * A) + (B * B)
   jta1 = Sqr(temp1)
End Function
```

- Alright, so can you put =jta1(3,4) in your spreadsheet and make it work?
- Yep, there are more tricks. It's *PRIVATE*. Make it *public* and try again.
- No? Try putting it into a Module rather than inside a sheet. Yep, it's weird.

Other Language Features and Details

- Like any language, VBA has *variables* and *constants*. Variables can be *local* (within the Sub or Function) or *global*.
  - Judiciously used, global variables are extremely powerful.
  - Named constants are your friend. Would you rather remember to type MODULE_ADDRESS everywhere or h3910FE9D?
- *Variables* are defined using the DIM statement.
  - As you'd expect, variables can be made into **ARRAYS**. Use them. Love them. They make great places to hold data before you copy it out to the worksheet at the end of the code.
  - VBA also supports the definition of *Types*. Kind of a special case, but allows you to make your own X.Y.Z structures.
- Functions and Subroutines can have *arguments*, which are also DIM'd to insure you match type and size

Other Language Features and Details

- Beware of different *variable* types
  - Long, Short, Integer, Double, String and (eeewwww...)Variant.
- The *Variant* type means “I don’t know what I’m doing so I have instructed the computer to pick the worst possible way to store this information at all times”.
- Converting values between types can be dangerous.
  - Biggest problems are trying to treat variables as binary bit things
  - Sign extension and/or odd logical results may occur
  - In short, it’s not ‘C’ and you shouldn’t assume the moral equivalents of <<, >>, &, # work exactly like you think they should.

Do's and Don'ts

- DO use the WITH statement (execution speed)
- DO learn the Object Browser (the F2 key)
- DO use Option Explicit
- DO use arrays and, where appropriate, custom structure types
- DO learn and use Application Functions
- DO NOT use the cells as variables (really slow)
- DO NOT get caught with signed/unsigned variable issues
- DO NOT fail to make a clean user interface your first priority
- DO NOT write routines other people have done before! 99.999% of the time it's already been done!